UCLouvain

LSINF2275 - Data Mining and Decision Making

Academic year 2019 – 2020

Project: Markov Decision Processes

Students:
Group 30
Dujardin Simon (57521400)
(DATI2MS/G)
Wynen Maxence (72971400)
(DATE2MS/G)

Professor:
Saerens Marco

Delivered on 22 March 2020

1 Introduction

In this project, we were asked to implement a Markov Decision Process algorithm to solve games of Snakes and Ladders. While the structure of the board remained the same for each game, layouts of this board (location for different traps) could change. The purpose of the algorithm was to find the optimal strategy of choice between two dices (a normal and a safe one) on every square of a given board. This document will explain the design choices we made in order to achieve this goal, the details of our implementation, and finally a discussion and a comparison with empirical results (i.e. thousands of simulations of the game).

2 Finding the optimal strategy

The model

As requested, we modeled our problem as a Markov Decision process. Basically the problem relies on choosing a dice at each turn so we acted as follow : each square of the board is represented as a state in the markov chain and at each state, choosing *D1* ("security" dice) or *D2* ("normal" dice) will lead to two different sets of transition probabilities to other states. Figure 1 illustrates such a model.

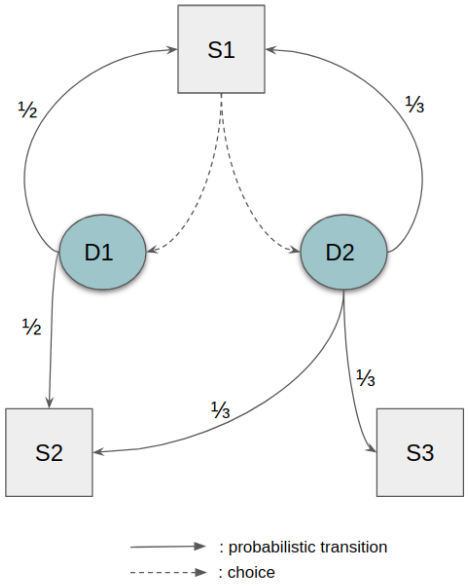

Figure 1: Markov diagram

So far we have not yet considered traps. To better explain what we did to manage them, here is an example : imagine the player is on square 3 and there is a *type-2* trap on square 4 (immediately teleports the player 3 squares backwards) as shown in the figure 2. Since using the *D1* is safe, making that choice would allow the player to move safely on the square 4 with a probability of 0.5. However for a choice of *D2*, the player has a 1/3 probability of landing on each of the safe squares accessible (S3 and S5) and a 1/3 probability of landing on the trap (S4) and thus on S1.

The modelling of the other traps is based on the same idea but the *type-3* (and so the *type-4*) is a bit more complex; it will thus be explain more in details section 3.

The computation

Once the model is built, we need to run an iterative method in order to find the best strategy. It was requested to use the "value iteration" method which is described at Figure 3. This method tends to find the strategy

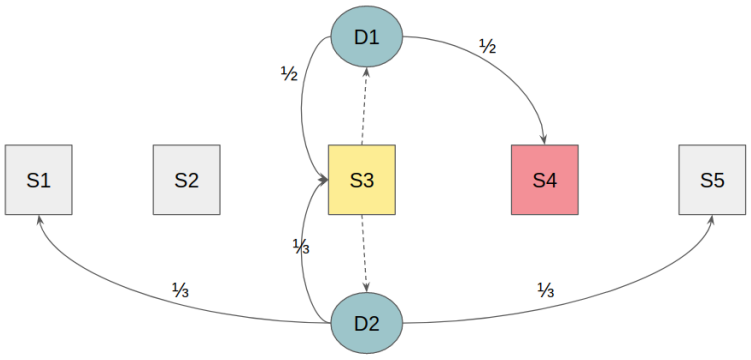

Figure 2: Trap diagram

$$\begin{cases} \hat{V}(k) \leftarrow \min\limits_{a \in U(k)} \left\{ c(a|k) + \sum\limits_{k'=1}^{n} p(k'|k,a)\hat{V}(k') \right\}, k \neq d \\ \hat{V}(d) \leftarrow 0, \text{ where } d \text{ is the destination state} \end{cases}$$

Figure 3: Value iteration formula

that minimizes its total cost, it is therefore trivial to make a relation between this cost function and the winning condition of the "Snakes and Ladders" game. The goal is to reach the last (15th) square of the board by playing the least possible times. Each action (of choosing a dice) has a cost of 1. By minimizing this cost function, the value iteration applied to the model described above should find the optimal strategy.

3 Implementation

Before doing the value iteration, we first had to build the transition matrix, which differs from layout to layout. This was made by the function `make_transition()`. In order to accelerate future accesses into this matrix, we have chosen to store this matrix as a dictionary where the states (i.e. squares and prisons states) are the keys. Each key (state) is associated to another dictionary of all actions available from this state (i.e. 'd1' for the security dice and 'd2' for the normal dice). Each action is then associated to a list of tuples of reachable states and the probability of landing on those states according to the chosen dice. All traps were easily implemented in this setup, except for the prison traps for which an explanation is needed. Before doing anything, our algorithm finds the number of prison traps as well as the squares they are at. For each prison trap, we added an associated state with identifiers that start at 16 (16 for the first prison, 17 for the second and so forth). If the trap is triggered, the player automatically goes to the associated "prison state". From this prison state, both actions have the same outcome of going back to the square containing the trap with probability of 1.0.

Once the transition matrix (which is actually a dictionnary) is correctly built, it is easy to write a function that does the "value iteration" algorithm on this matrix (see our function `value_iteration()`). Globally our function `markovDecision()` follows these few steps :

- Build the transition matrix
- Initialize the data structures `states` and `reward` (that corresponds to the cost function mentioned above, i.e. 1 for each square, execpt for the 15th)
- Execute the 'value iteration'

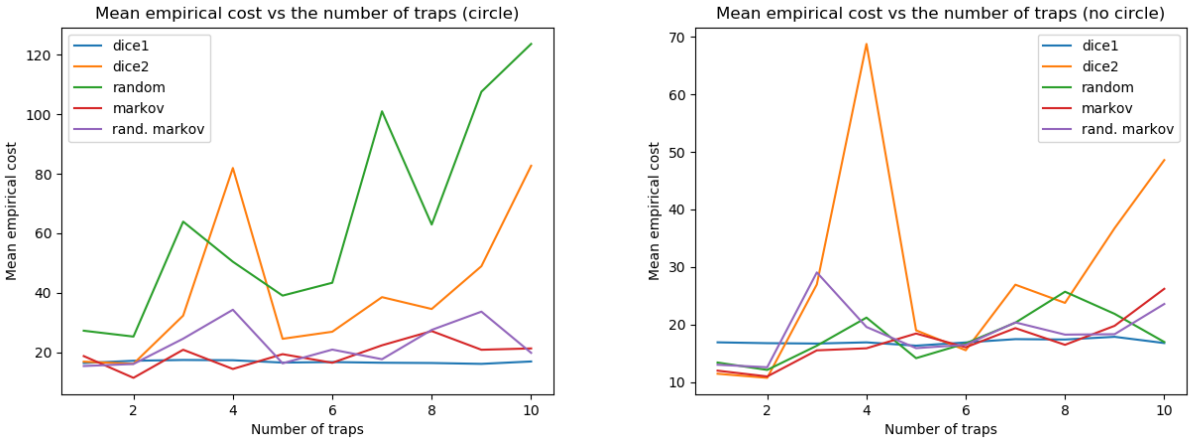

Figure 4: Evolution of the empirical cost means according to the number of traps (circle and no circle)

- Extract the best strategy from the results of the previous step
- Convert the results in the requested format and return them

4 Comparison with the empirical results and Discussion

For this part, we have chosen to work on 21 random layouts with an increasing number of traps (from 0 to 10). All layouts can be found in Appendix A. To have a better view of the performance of the strategy our algorithm has found, we decided to compare it to several strategies:

- *Dice 1* : Only use dice 1, the 'safe' dice
- *Dice 2* : Only use dice 2, the 'normal' dice
- *Random* : Choose at random between the normal and the safe dice
- *Random Markov* : choose the 'markov' dice with a probability of 0.8 and the other choice with a probability of 0.2

These strategies should give us a better insight at how well our algorithm works. Obviously, we expect it to show the lowest cost. Let's see if this is the case. The simulation was made on each layout a thousand times for the circle and the non circle modes, and results of the actual cost can be found on the following Figures 5 and 4.

Let's first take a glance at Figure 4, showing the evolution of the different strategies' mean in terms of the number of traps. The first thing to note is that *Dice 1* is almost a straight line, which is logical since it cannot trigger traps. The next strategy we see is *Dice 2*, which cannot avoid traps. Without much surprise, this strategy gives poor results in general. In the circle mode, both *Dice 2* and *Random* give very poor results as they very easily go past the 15th square and have to start all over again. Other strategies don't seem too affected by the number or nature of the traps, but more by their location.

Let's now see what the empirical cost distribution is for each strategy (Figure 5. Note that these results come from the same layouts used in Figure 4, so they summarize 21 000 simulations of the game.
Regarding the circle mode first, we immediately see that the worst strategies are *Random* and *Dice 2*, for reasons explained before. To have a clearer view at the same results, let's take a look at the same graph without the *Random* strategy (Figure 6). *Dice 2* can both give low and very high results, but is in general worse than other strategies (except *Random*). *Dice 1* is nearly the same in circle and no circle modes, as it can

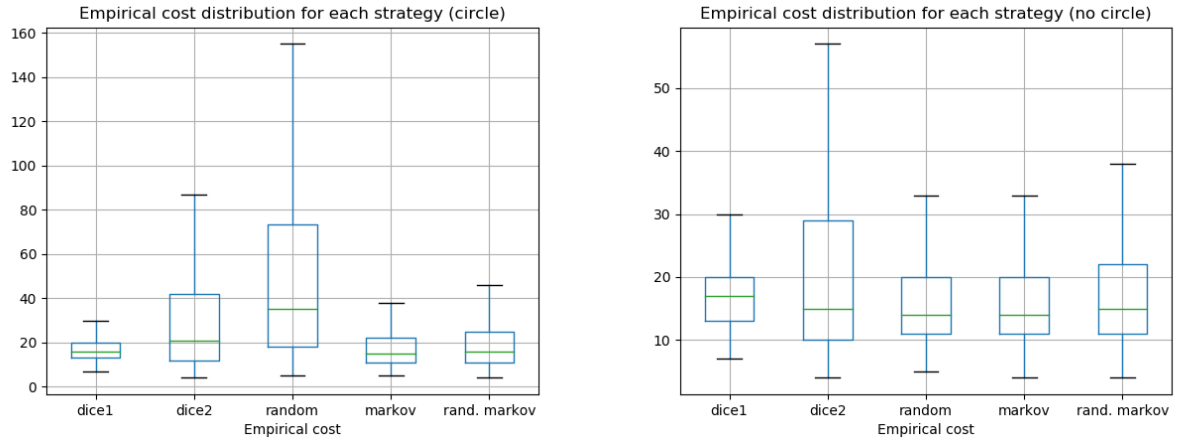

Figure 5: Empirical cost distribution for each strategy (circle and no circle)

never exceed the 15th square. We also observe the range of *Dice 1* being the shortest. Without much surprise, *Random Markov* gives slightly worse results than *Markov*. In summary, *Markov* gives generally speaking the best results.
For the no circle mode, *Dice 2* is clearly the worst strategy, as the cost can be very high. Furthermore, *Random* and *Markov* look oddly very similar (although *Markov*'s boxplot is slightly lower) which tells us that *Random* might be a good strategy in this mode. As previously mentioned, *Dice 1* gives approximately the same results as in the circle mode. Its median is however significantly higher than *Markov*'s. Once more, *Random Markov* gives slightly worse results than *Markov*. In short, *Markov* seems indeed the best way to go if one tries to minimize the number of turns.

Finally, Table 1 displays a comparison between the expected costs and the empirical means for each layout. We observe a mean difference between the empirical means and the expected costs of 6.2 in circle mode and 4.3 in no circle mode. This (big) difference could potentially be explained by a slight dysfunction in our algorithm that we could have overlooked.

5 Conclusion

The purpose of this assignment was to write a Markov Decision Process applied on a game of Snakes and Ladders with a given board, and movable traps. This algorithm had to find the optimal strategy, i.e. linking each square to a choice of dice. The whereabouts of the game were explained in Section 2. We chose to implement the Value Iteration algorithm on a specific data structure. Both were expanded in Section 3. Finally, Section 4 compared different strategies with our optimal one on several random layouts with different number of traps. This showed that the strategy found by our function was the way to follow, as it statistically gave the best empirical results. Let's note the particularly poor results of the *Random* strategy in the circle mode as well as the extroardinary similarity between our optimal strategy and the *Random* one when the circle mode is not activated. Further analysis can be found in this section.
In short, this project got us to be way more familiar with the Markov Decision Processes. It also gave us a great opportunity to compare probabilities with actual statistical results.

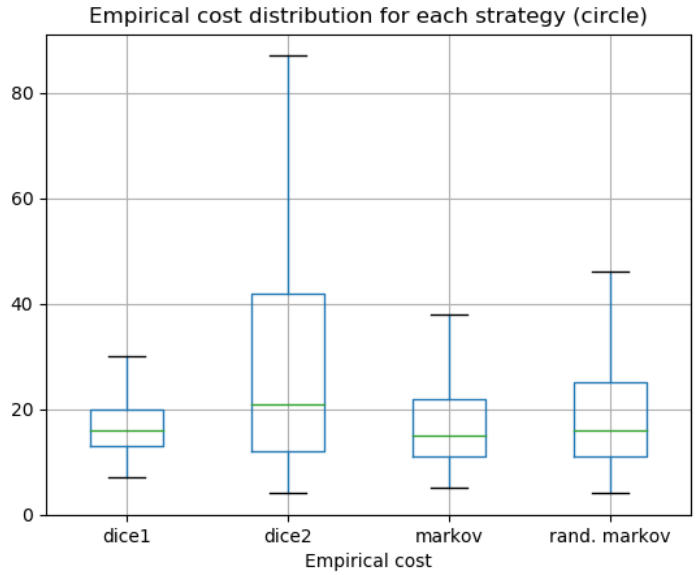

Figure 6: Empirical cost distribution for all strategies except *Random* in circle mode

	Circle			No Circle		
Layout n	Expected	Empirical	Emp-Exp	Expected	Empirical	Emp-Exp
1	11,09	25,97	14,88	10,73	13,71	2,98
2	10,76	11,51	0,75	10,44	10,31	-0,13
3	10,6	11,29	0,69	10,2	11,31	1,11
4	10,33	11,57	1,24	9,98	10,7	0,72
5	12,49	26,68	14,19	12,31	16,38	4,07
6	12,8	15,06	2,26	12,8	14,7	1,9
7	12,47	13,16	0,69	12,47	13,37	0,9
8	13,56	15,67	2,11	13,43	18,42	4,99
9	12,62	19,61	6,99	12,43	15,2	2,77
10	12,59	19,16	6,57	12,26	21,72	9,46
11	12,71	17,33	4,62	12,3	15,98	3,68
12	14,59	15,66	1,07	14,17	16,16	1,99
13	15,06	21,79	6,73	14,65	16,56	1,91
14	12,18	22,94	10,76	11,84	22,23	10,39
15	12,71	32,45	19,74	12,54	17,18	4,64
16	15,31	21,85	6,54	14,9	15,82	0,92
17	14,89	24,3	9,41	14,74	23,01	8,27
18	14,49	17,44	2,95	14,24	16,59	2,35
19	15,49	25,1	9,61	14,99	35,46	20,47
20	14,74	17,53	2,79	14,31	16,99	2,68
Mean	**13,074**	**19,3035**	**6,2295**	**12,7865**	**17,09**	**4,3035**

Table 1: Expected costs vs Empirical mean costs

Appendix

A Layouts used for the empirical results

Layout 0 : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Layout 1 : [0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0]
Layout 2 : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0]
Layout 3 : [0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
Layout 4 : [0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0]
Layout 5 : [0, 0, 0, 0, 0, 1, 4, 0, 0, 0, 0, 0, 0, 1, 0]
Layout 6 : [0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 2, 0, 1, 0]
Layout 7 : [0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 4, 4, 0]
Layout 8 : [0, 1, 0, 4, 0, 0, 0, 0, 0, 2, 0, 4, 0, 0, 0]
Layout 9 : [0, 1, 0, 0, 0, 0, 4, 0, 3, 0, 3, 0, 0, 2, 0]
Layout 10: [4, 3, 0, 0, 3, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]
Layout 11: [0, 1, 3, 4, 1, 0, 0, 0, 0, 0, 4, 0, 0, 3, 0]
Layout 12: [1, 3, 4, 0, 4, 3, 0, 0, 0, 0, 0, 0, 2, 0, 0]
Layout 13: [0, 0, 4, 0, 3, 1, 4, 2, 2, 0, 0, 0, 0, 3, 0]
Layout 14: [0, 0, 3, 1, 1, 2, 1, 3, 0, 4, 0, 0, 0, 0, 0]
Layout 15: [2, 1, 3, 0, 3, 0, 4, 0, 3, 1, 0, 0, 0, 3, 0]
Layout 16: [0, 4, 4, 1, 1, 0, 1, 3, 2, 0, 0, 0, 3, 0, 0]
Layout 17: [0, 2, 0, 2, 3, 0, 2, 3, 1, 1, 3, 1, 0, 0, 0]
Layout 18: [0, 1, 3, 0, 3, 2, 4, 1, 2, 4, 4, 0, 0, 0, 0]
Layout 19: [4, 1, 3, 2, 2, 2, 3, 0, 2, 0, 3, 0, 1, 0, 0]
Layout 20: [2, 2, 0, 4, 4, 4, 3, 4, 3, 0, 0, 3, 4, 0, 0]